Vorlesungsreihe „Diskrete Simulation“ (Masterkurs)

Verteilte diskrete Simulation und entsprechende Standardisierungssansätze

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Überblick

- **Grundlagen der verteilten diskreten Simulation**
  - Motivation und Ziele
  - Historische Entwicklung
  - Arten der verteilten Simulation
  - Zeitsteuerung bei der verteilten parallelen Simulation
    - Konservativer Ansatz
    - Optimistischer Ansatz

- **Technologien und Standards zur verteilten Simulation**
  - Historie und Ausgangsbasis bei der Entwicklung
  - Die High-Level-Architecture (HLA) des DoD
  - Neue Technologien und Interoperabilitätsansätze

Anforderungen an verteilte diskrete Simulationen

Performanceanforderungen

- sehr große Modelle können bei den Laufzeiten zu inakzeptablen Werten führen
- eine aufsetzende Optimierung kann auch bei kleinen Einzellaufzeiten zu größeren Gesamtuntersuchungsdauern führen

Interaktion/ Kopplung verschiedener Simulationsmodelle und –systeme (Interop):

- in sehr komplexen Szenarien müssen mehrere, ggf. auch historisch gewachsene Simulationsmodelle miteinander interagieren
- Typische Problemfelder sind
  - Militärische Komplexsimulationen mit getrennten Modellen für Land / See / Luft und tw. integrierten Realtime-Simulatoren (Flugsimulator mit Pilot o.ä.)
  - Supply-Chain-Management-Simulationen mit unterschiedlichen Simulations-modellen/-systemen bei Auftraggeber und verschiedenen OEM´s

Weitere Anforderungen

- Parallelausführung zur Fehlerkontrolle (Null-Fehler-Anforderung)
- Räumlich /geografisch verteilte Modelle (Niederlassungen globaler Konzerne)
- früher auch wegen Speicherplatzbegrenzungen (heute obsolet)

Historische Entwicklung

Erste Ansätze bereits in den 70er Jahren:

- Mit Verbreitung der EDV vor allen im militärischen Bereich auch neue Ansprüche an die Komplexität von militärischen Simulationen (Planspiele)
- Taktfrequenz der Rechner noch sehr gering (geringer MHz-Bereich)
- damit Motivation zur verteilten Simulation zur Beschleunigung der Berechnungen

Einbindung der PC-Technik in den 80er/ 90 er Jahren

- PC´s als intelligente Bedien-Frontends für große Simulationsmodelle im militärischen Bereich
- Verteilte Simulation auf der Basis auch von PC´s durch aufkommende Vernetzung
- großes Projekt SIMNET (SIMulator NETworking) von 1983 bis 1996

Arten der verteilten Simulation

Nach Fujimoto (Fuji99) werden unterschieden:

1. Verteilte ANALYTISCHE Simulationen:

- Hauptziel ist die (statistische) Analyse der Modelleigenschaften
- Geschwindigkeit: **SO SCHNELL WIE MÖGLICH (abs. Wert kann variieren)**
- keine größeren Interaktionen (ggf. auch Batch-Betrieb ohne Oberflächen)

2. Verteilte Virtuelle Umgebungen (Distributed Virtual Environments)

- Hauptziel ist die Schaffung künstlicher Umgebungen zur quasi-realistischen Schulung oder dem Test möglicher Einsatzszenarien meist unter Einbeziehung von menschlichen Akteuren
- Geschwindigkeit: **GENAU WIE REAL-Zeit (Verhältnis zur Realzeit sollte konstant sein -> ggf. muss die Simulation ausgebremst werden)**
- große Ähnlichkeit und Nachnutzung von Technologien aus dem Spielebereich (tw. auch Gaming-Frameworks wie z.B. Havok-Engine)

Verfügbare Hardware für die verteilte Simulation

1. (Echte) Parallelrechner (Supercomputer):

- spezialisierte Hardware mit tw. mehreren Computern / Prozessoren pro Platine und gemeinsamen Speicherbereichen (meist sehr teuer)
- Vorteil(e)(e): Datenaustausch zw. Rechnern sehr schnell
- Nachteile: meist auch spezialisierte Betriebssysteme und damit Standard-Simulationssysteme nicht einsetzbar (-> Programmierung mit C/C++ u.a.)

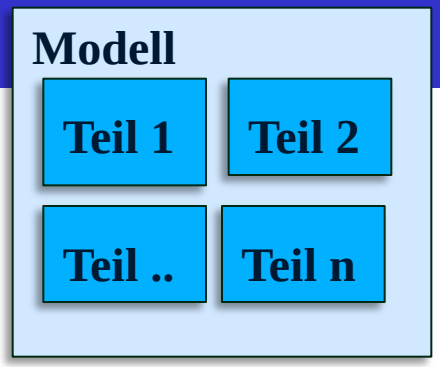
2. Vernetzte Standard-(Personal)-Computer

- Standardrechner, ggf. auch mit Mehrprozessorsystemen
- Kopplung der Rechner über Standardnetzwerke (100MBit/..GBit)
- Vorteile: relativ kostengünstig, Nutzung der Rechner in Leerlaufzeiten (Nachts/Wochenende)
- Nachteile: Kopplung über Netzwerk relativ langsam

3. NEU: Standardrechner (PC) mit spezialisierten Grafikkarten

- Kombinieren die Vorteile beider Optionen (Verfügbarkeit/ Kosten)
- Heutige Grafikkarten noch eingeschränkt bei der Flexibilität

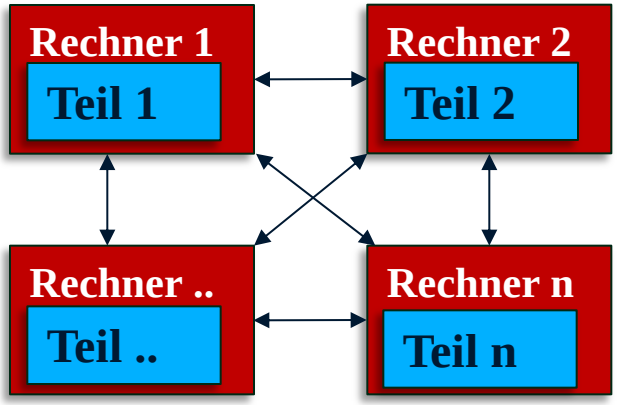
Arten der verteilten Simulation



Nach Fujimoto (Fuji99) werden wiederum unterschieden:

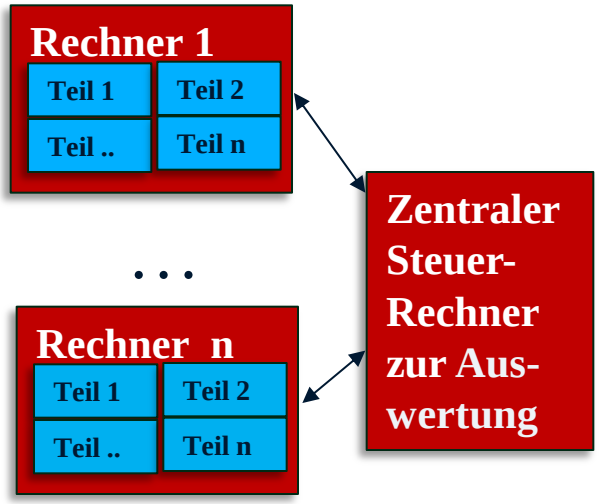
1. **Parallel-Simulation:**

- Aufteilung des Modells (Partitionierung)
- jedes Teilmodell auf getrenntem Proz. / Rechner
- Kommunikation der Teilmodelle zum Austausch von Statusinformationen und ggf. auch dyna-mischen Objekten
- Hauptproblem: Synchronisation der Zeit



2. **Hybridsimulation**

- Verteilung kompletter Modelle auf jeweils EINEN Rechner, mit jeweils einer Replikation (Simulations-lauf) pro Rechner
- Keine direkte Kommunikation zischen den den Modellen, nur Übermittlung der Ergebnisse an einen zentralen Steuer- und Ergebnisrechner



Bewertung der Effizienz mit dem Speedup-Koeffizienten

Die Effizienz einer verteilten Simulation kann gemessen werden mit dem:

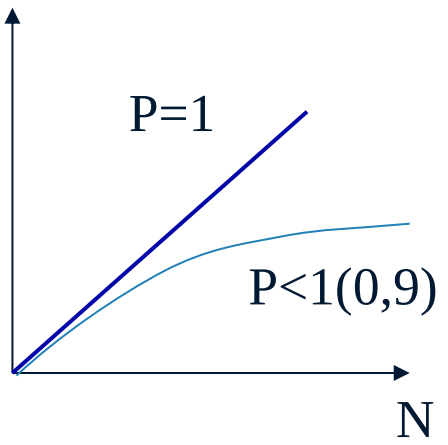
Speedup-Koeffizienten S_p:

$$S_p = \frac{T_1}{T_p}$$

- Verhältnis zwischen der Zeit T_1 für die Berechnung auf einem Prozessor und bei paralleler Berechnung T_p mit N-Prozessoren
- theoretisch kann der Koeffizient über das Amdahlsche Gesetz (1967) berechnet werden mit
  - P- der parallelisierbare Anteil
  - N ist die Anzahl der Prozessoren

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

- Bei P=100% (selten erreichbar) ergibt sich S= N !
- In der Praxis sind Werte von P= 70 bis 95% real, da immer nicht-parallelisierbare Teile existieren oder Kommunikationen zwischen Komponenten nötig sind.
- Bei der Hybridsimulation kann ein P nahe bei 1 erreicht werden, da nur Ergebnisse OHNE Warten gespeichert oder gesendet werden. Damit ist der Speedup meist nahe bei N !

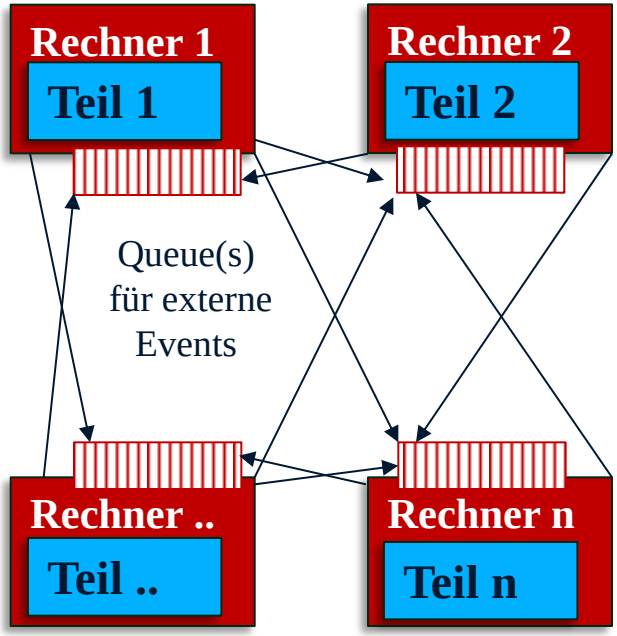


Zeitsynchronisation bei der parallelen Simulation

Durch die Zerlegung und Aufteilung des Modells auf mehrere Prozessoren laufen diese getrennt und müssen synchronisiert werden. -> Hauptaufwand bei der vert. Simulation zur Absicherung der logischen Korrektheit/Kausalität

1. Konservativer Ansatz:

- Jedes Teilmodell (auch Logischer Prozess (LP) in der Literatur genannt) sendet Nachrichten mit Timestamps aus, welche die nächste eigene Uhrzeit,(dies ist gleichzeitig auch ein „Versprechen“ kein niedrigeren Zeiten zu senden).
- Jeder LP verarbeitet die einkommenden Timestamps und seine eigenen Ereignisse zeitlich sortiert
- Bei grossen Lücken werden auch sogen. Null-Messages gesandet (KEIN direktes Ereignis, nur reine Zeitfortschaltung)
- Intervall zw. 2 Null-Messages ist der sogen. **Lookahead:**
  - Kleiner Wert (z.B. 1s) -> viele Nachrichten, aber sicher
  - grosser Wert (10s) -> eventuell Inkonsist. & Deadlock´s
- **Nachteile: hohes Nachrichtenaufkommen durch 1 zu M-Aussendung jeder Msg an alle LP, ggf. starkes Ausbremsen durch langsame Teilprozesse - der langsamste Prozess bestimmt IMMER die Gesamtperformance !**



Optimistische Zeitsynchronisation

2. Optimistischer Ansatz:

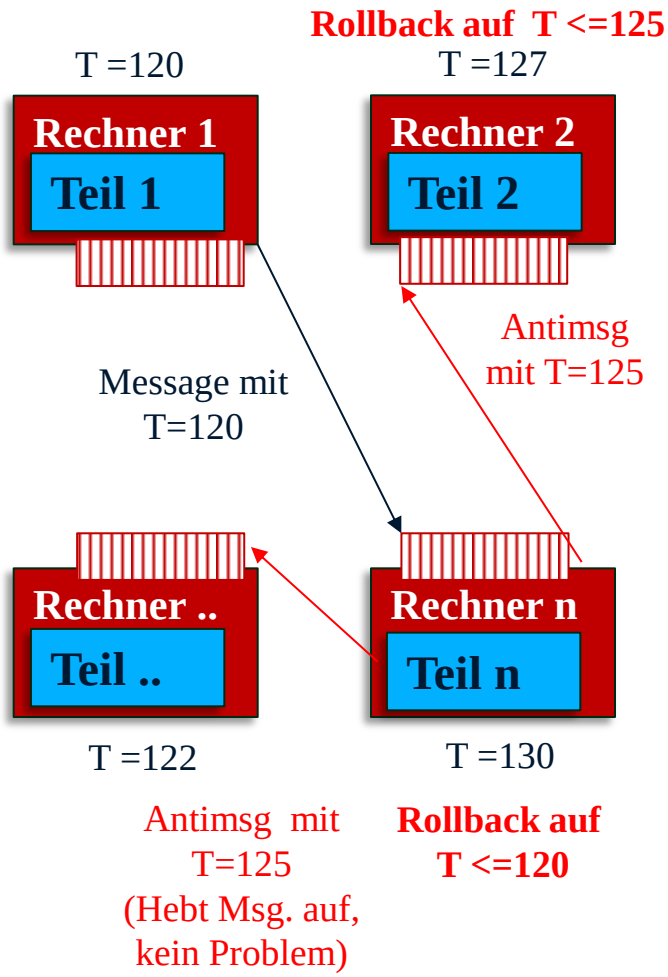
- Im Gegensatz zur konservativen Methode, welche Inkonsistenzen im Gesamtmodell generell vermeiden will, lässt der optimistische Ansatz Fehler bei der Kausalität zu und versucht diese zu beheben.

Folgen:

- LP´s können der globalen Zeit (GT) VORAUS eilen.
- Es werden nur echte Interaktionen (z.B. Produkt-transfer) zw. LP´s ausgesendet (also deutlich weniger!!)
- Damit können aber auch Messages mit Timestamps aus der „Vergangenheit“ eintreffen !

Lösung mit „TimeWarp“ (nach Fujimoto / (Jefferson 1985):

- Im zu weit voraus geeilten LP wird ein Rollback auf die zurückliegende Zeit ausgeführt (z.B. auf der Basis von Snapshots oder Transaktionslog {analog DB´s}).
- Dabei rückgängig zu machende Messsageaussendungen werden mit Antimessages ungültig gemacht.
- Sind Messsags aber vor Eintreffen der Antimessages in anderen LP´s verarbeitet, müssen auch dort Rollbacks durchgeführt werden (-> ggf. Rollback-Kaskade (;-((!)



Optimistische Zeitsynchronisation II

Der Optimistische Ansatz ist besonders geeignet bei

- **Pulkartigen Operationen** (z.B. Batchaufteilung oder Gruppenankünften z.B. am Flughafengate) innerhalb von Modellteilen OHNE Interaktionen mit anderen Modellteilen:
  - Ein Modellteil bleibt etwas zurück und holt dann aber wieder auf
  - andere Teile eilen voraus und bekommen dann aber ebenfalls mehr „Arbeit“
  - In der Summe gleichen sich die Modellteile immer wieder an und sind in Summe schneller, als wenn jeweils der langsamste das Tempo vorgibt.
  - Sinnvoll ist eine Begrenzung des Vorauseilens (durch Sliding Time Windows oder modellspezifische Vorgaben), um die Anzahl von Rollback zu begrenzen

Der Optimistische Ansatz ist NICHT geeignet bei

- generell starken Laufzeitunterschieden der Teilmodelle, da dann immer wieder Rollbacks ohne Gesamtlaufzeitgewinn stattfinden,
- sehr fein gegliederten Simulationen mit fast quasi-kontinuierlichen Zeitverhalten und/oder starker Modellteilinteraktionen (weil dann sehr viele Zeitstempel-Messages versandt werden müssen)

Konvervative vs. Optimistische Synchronisation

Generell

- gibt es **KEINE eindeutige Kriterien** für die Verwendung des jeweiligen Ansatzes
- Die konkreten Vor- und Nachteile der beiden Ansätze werden i.d.R. häufig überlagert von der Qualität der Modellpartionierung: durch eine sehr gute Partionierung mit sehr wenigen Modellteilinteraktionen kann sich das Laufzeitverhalten speziell des optimistischen Ansatzes sehr deutlich verbessern.

Der Konservative Ansatz

- ist **einfacher zu implementieren**, da nur das Messagings-System zum Austausch der Zeitstempel zu existierenden Simulationssteuerungen hinzugefügt werden muss
- Wenn ein „guter“ Lookahead ermittelbar ist, sind konserv. Ansätze relativ günstig.
- benötigt möglichst schnelle Netzwerke zum Message-Austausch

Der Optimistische Ansatz

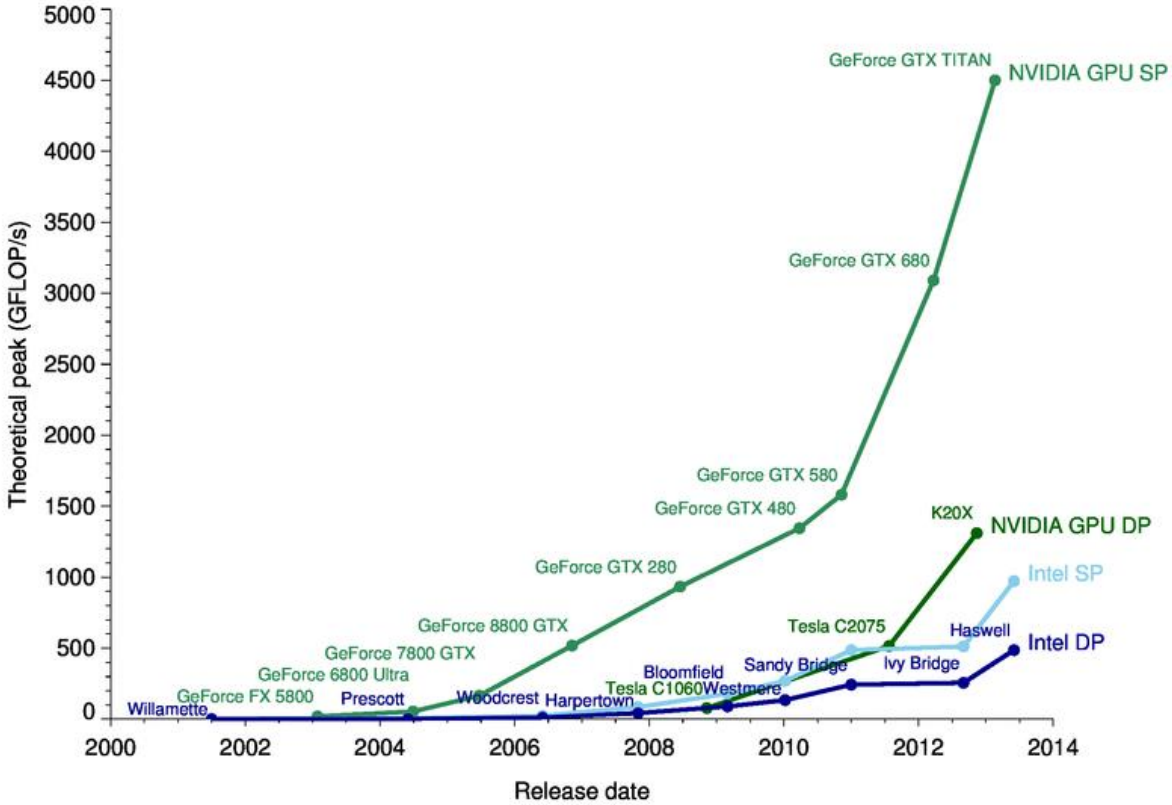
- erfordert **größere Entwicklungsaufwendungen** durch die Notwendigkeit der Rollbacks tw. sehr aufwändige Zwischenspeicherungen (Snapshots) und Rückholmechanismen, welche sehr stark in das Modell eingreifen und führt zu stark spezialisierten Sim.-Systemen

In der Praxis entscheidet häufig die konkrete Verfügbarkeit über den Einsatz !

Neue Option : Grid / GPU-basierte Berechnungen

- stille Revolution im Bereich der Grafikkartenleistung
- -> GPU-Leistung mehrfach höher als traditionelle CPU´s
- Hersteller NIVDIA and AMD (ATI) -> CUDA / OpenCL-Standards !

➔ **Frage : Ist dies für Simulationszwecke nutzbar ?**



Quelle der Grafik : https://michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html

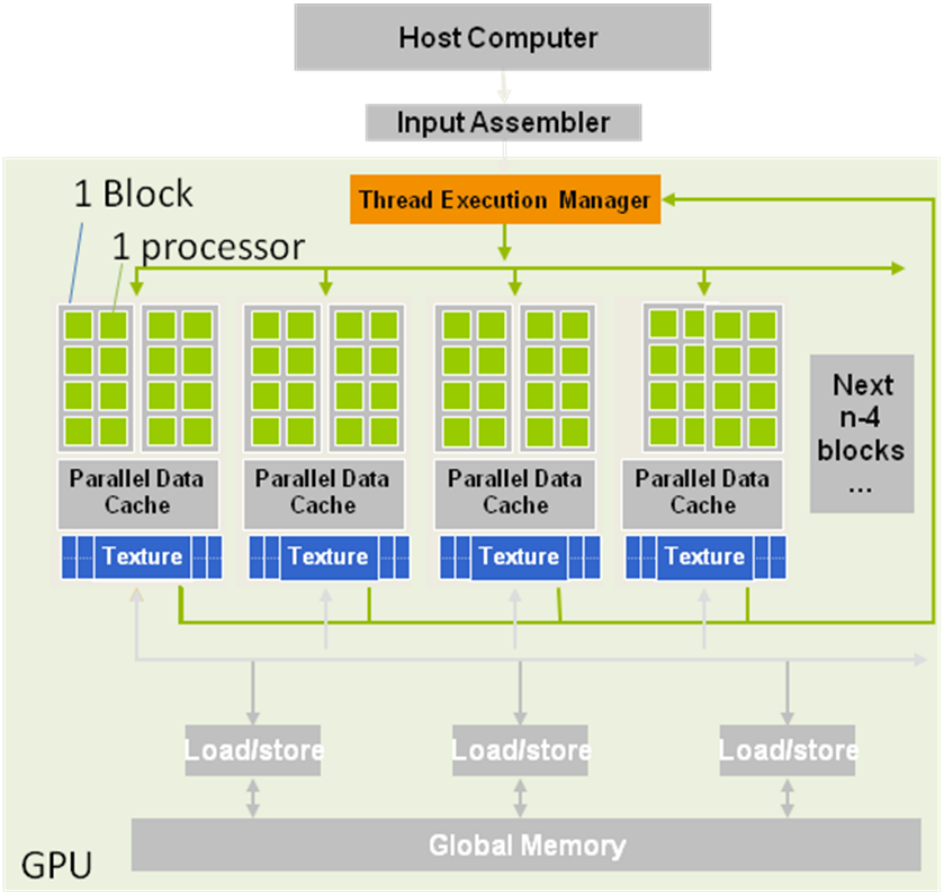
Graphic Processor Units (GPU) – Architecture

Ausgangspunkt:

- 3D-Szenarien erfordern sehr viele Shader and Texture-Prozessoren
- aktuell 1024 und mehr Prozessoren mit floating (double) point prec.

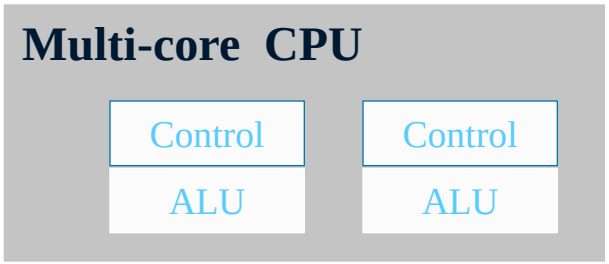
Memory - Organization :

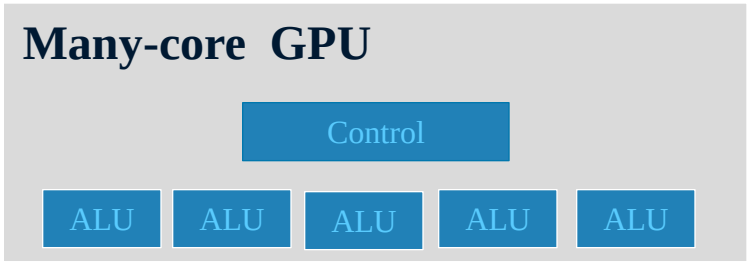
- ist der limitierende Faktor !!!
- Schnellster Speicher ist shared memory in den Blöcken
- nur zwischen diesen Speicherzellen ist ein sehr schneller Datenaustausch möglich
- Synchronization zwischen den Blöcken ist langsam und schlecht unterstützt !



Problem : Single Program Multiple-Data (SPMD)

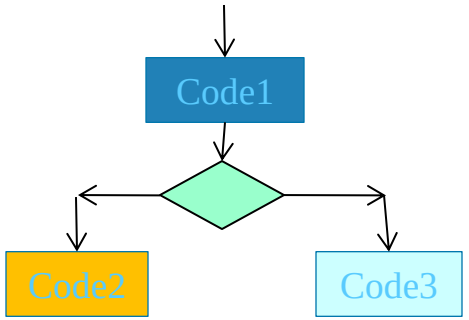
- in CPU-Proz. existiert ein Control block per CPU.
- in GPU´s exisitiert NUR ein Control block für N-ALU´s ! !!!

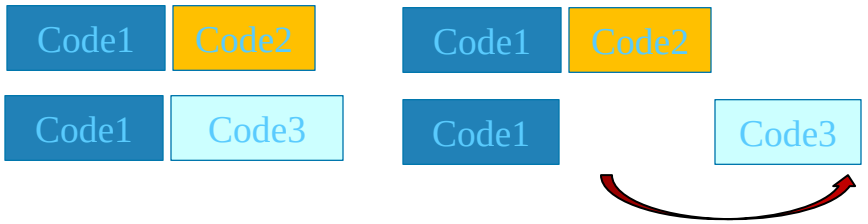




Folgen:

- **nur 1 Program kann über N-Daten ausgeführt werden**
- bei Sprüngen im Code kommt es dadurch zur einer SEQUENTIELLEN Ausführung mit drastischen Performanceverlusten





Mögliche Anwendungen von GPU´s im Simulationsbereich

Parallel & Hypercomputing mit unters. Zufallszahlenströmen

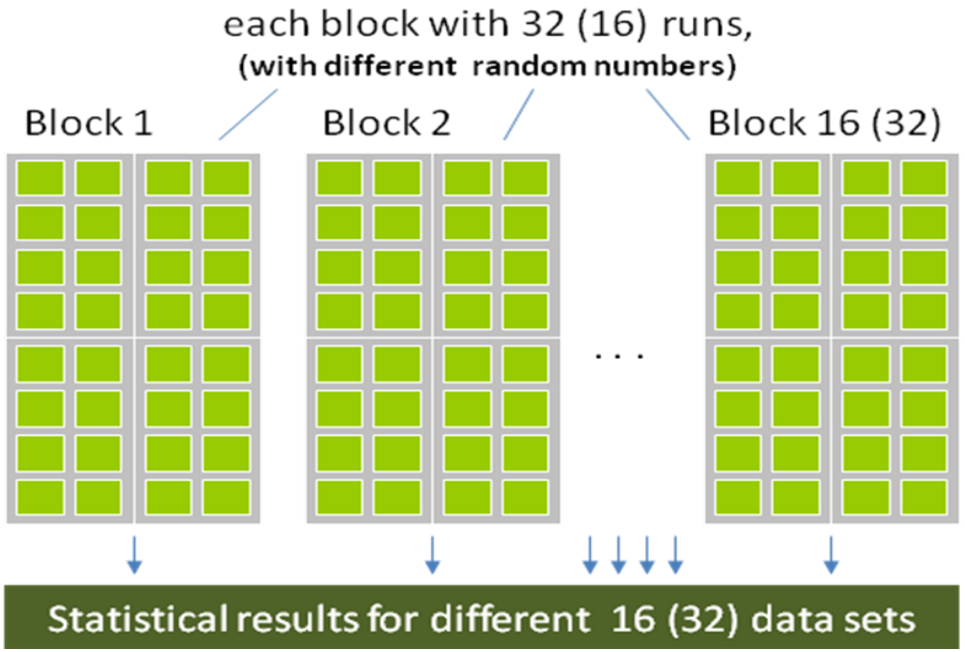
- Monte-Carlo-Simulationen
- einfache Modelle OHNE Verzweigung infolge zufälliger Ereignisse

GPU -Solution : M-Modellläufe mit N-Szenarien

- pro Block ein Lauf mit N (=kernel-Anzahl)-Einzelsimulationen
- Unterschiedliche Szenarien in den Blocks (aber immer gleiche Codeausf.!)

Restriktionen /Ergebnisse :

- pro Block gleicher Code
  und gleiche Ausführung !!!
- dann möglicher Speedup:
  - 512 …1024 (=Anzahl Kernel)



Aktuelles Fazit GPU-Einsatz:

- **Zur Zeit noch limitiert**
- **Aber weiter beobachten bzgl. Verbesserungen der Hardware und weniger Restriktionen !**

Konkrete Technologien und Standards - Historie

1. SimNET – Entwicklung

- Real-Time wide area network-Simulation des US-Militärs für Trainingszwecke
  - Hauptziel: billiger und sicherer als echte Manöver
  - Start 1980, erster Prototyp 1985, Feldeinsatz ab 1987
  - echte Bewährung als Trainings- und Planungsinstrument im 1. Golfkrieg
- bei Vollausbau in 1990 ca. 250 gekoppelte Simulatoren in 11 Sites mit
  - **Fahrzeug-Simulatoren (Army)**
  - **Flugsimulatoren (Airforce)**
  - jeweils mit Real-World-Einheiten (echte Panzer), virtuellen Trainingsgeräten und rein simulativen Einheiten

Neue technologische Ansätze

- Datenaustausch mit den anderen Simulatoren über Protocol data units (PDUs), teilweise zu Beginn stark komprimiert über 56k-Modems (auch mit Voice)
- **Dead Reckoning** Algorithmen zur Berechnung der Bewegung mobiler Einheiten über deren PDUs mit Ausgangsposition & Bewegungsvektor
  - nur bei stärkerer Abweichung von berechneter Position erfolgt Update
  - dies reduziert die Positionsupdates sehr stark

2. DIS – Distributed Interactive Simulation

- Nachfolgestandard zu SimNET mit starker Anlehnung an diesen,
- zuerst wieder im MIL-Bereich, später auch in der Logistik, Medizin und Raumfahrt
- definiert als IEEE-Standard 1278 und umfasst folgende Standardgruppen
  - Application Protocols, Communication Services and Profiles
  - Exercise Management and Feedback, Verification Validation & Accreditation
- Die DIS-Application Protocols definieren APU´s (konkrete MSG-Formate) u.a. für:
  - Entity information/interaction family - Entity State, Collision, …
  - Warfare family - Fire, Detonation, Entity Damage Status
  - Logistics family - Service / Repair Operations
  - Simulation management family - Start/Resume, Stop/Freeze, Acknowledge
  - Radio communications family - Transmitter, Signal ...
  - Information Operations family - Information Operations

- Die Entwicklung und Betreuung des Standards erfolgte durch die **SISO** (Simulation Interoperability Standards Organization), welche auch heute noch aktiv ist,
- SISO-Website unter http://sisostds.org/

3. HLA – High Level Architecture

- Nachfolgestandard zu DIS, unter Nachnutzung vieler Komponenten
- erster Entwurf 1996, volle praktische Inbetriebnahme ab 1998/99
- zuerst vorangetrieben vom Pentagon (DoD) zuerst wieder im MIL-Bereich, später auch explizit zur Nutzung in zivilen Bereichen (Logistik, Kommunikation, …)
- definiert als IEEE-Standard 1516

Hauptziele im Vergleich zu SImNET und DIS

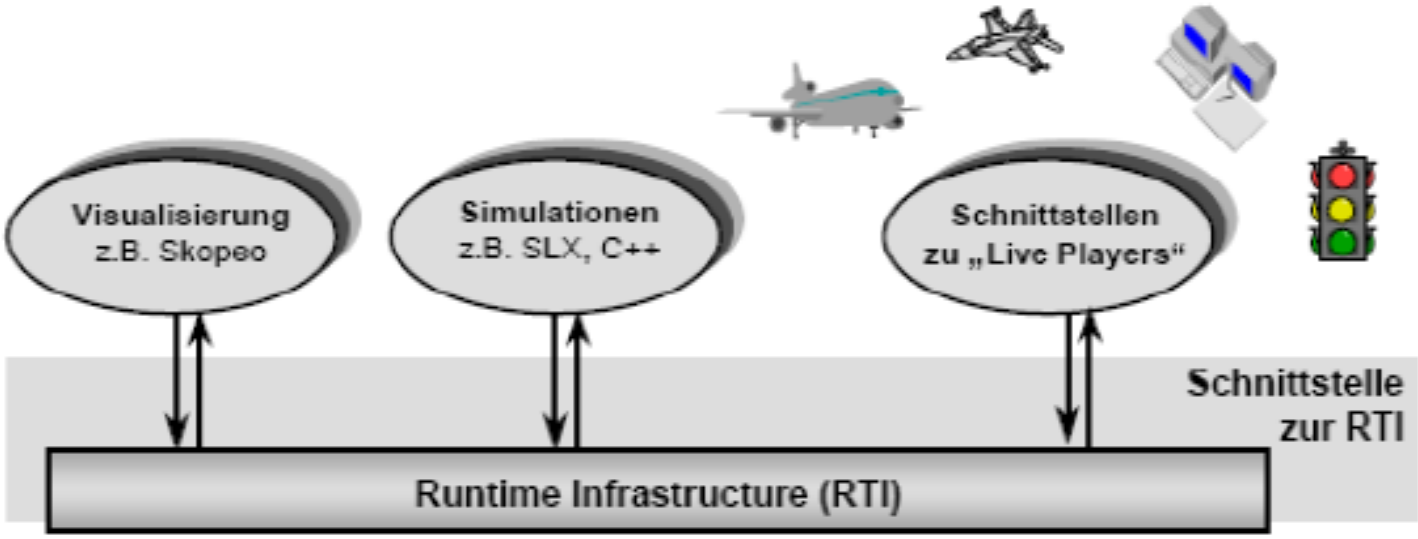
- bessere Wiederverwendbarkeit aller Technologien
- bessere Interoperabilität der eingesetzten Simulatoren (Kopplung von Simulatoren statt immer wieder neue Kopplung von Simulationsmodellen)
- allgemeine Verbesserung der bisherigen Technologien im Bereich der verteilten Berechnung

- Die Entwicklung und Betreuung des Standards erfolgte wieder durch die **SISO**
  SISO-Website unter http://sisostds.org/

HLA - Komponenten

Eine High Level Architecture (HLA) besteht aus

- einer HLA-Runtime-Infrastructure (RTI) , welche die gesamten Daten-kommunikation abwickelt (heute vergleichbar mit einem Servicebus)
- einer beliebigen Menge von Federates, welche i.d.R. Simulatoren oder Real-Time-Systeme darstellen, die Gesamtheit aller Federates bildet die Federation
- Die Federates und die RTI unterliegen den gleichen HLA-Regeln (HLA-Rules).



HLA - Definitionen

HLA definiert sehr genau

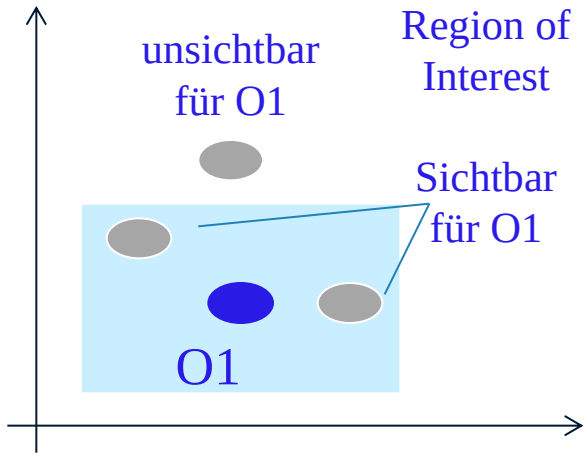
- **Die Interface Specification**
  - Objekt-orientierte Referenzspezifikationen für C++ and Java programming (in der ersten HLA-Version 1.3 auch für Ada and FORTRAN)
- **Ein Object Model Template (OMT)**
  - legt Templates (shared object, attributes and interactions) für die HLA - Datenobjekte fest
    - **Federation object model (FOM)** – Def. der gesamten Federation.
    - **Simulation object model (SOM)** – Def. eines einzelnen Federates.
  - auch Definition von Dokumentationsrichtlinien des OMT
- **Die HLA Rules**
  - 10 Verhaltensregeln für Federates & Federations
  - Regeln zur minimalen Datenpräsentation in den einzelnen Federates (z.B. muss die lokale Zeit eines Federates jederzeit abrufbar sein)
  - Laufzeitregeln zur Steuerung und Überwachung durch RTI
  - Dokumentationsrichtlinien zur gesamten Federation

Data Distribution (DD) in the HLA

- bei DIS : Information Broadcast an alle Teilnehmer
- mit steigender Modellgröße trotz Dead Reckoning- Alg. Probleme durch Versand (Bandbreite) und Verarbeitung der Nachrichten

Neues HLA-Konzept:

- Federates abonnieren nur interessierende Nachrichten
- bei mobilen Einheiten (und den typischen Manöverszenarien) wird eine **Region of Interest** als Funktion der eigenen Position definiert (z.B. als Funktion der Sichtweite optisch/ Funkwellen)
- je nach Szenario auch 3-dimensional
- weitere Unterteilung der Nachrichten in Subscription- und Updateareas auch mit unterschiedlichen Abmessungen
  (Panzer "sieht" im Gelände weniger als Hubschrauber)
- Balance der Area-Größen beeinflusst Performance:
  - geringe Größe = gute Filterung, aber häufige Regionwechsel mit Überlappungsprüfungen,
- - größere Bereiche = weniger Region-Wechsel, aber mehr Nachrichten



HLA – Weitere technologische Verbesserungen

Zeitliche Selbständigkeit der Simulatoren

- Aufgrund des RealTime-Szenarios sind die Simulatoren selbst für die Einhaltung der Kausalitätsbedingungen verantwortlich (immer mindestens so schnell wie Realtime)
- sowohl konservative wie optimsitische Synchronsisierungen werden unterstützt

Effiziente Datenupdates

- In der Regel nur Updates bei Änderungen des Verhaltens (über konstante Bewegungen wird nicht ständig benachrichigt)
- ggf. aber „Keep alive“ – Nachrihcten in größeren Abständen (5s), falls neue Federates während der Simulaitons hinzukommen
- Verbesserte Dead Reckoning Algorithmen *:*
  - *Sowohl Sender wie Empfänger berechnen nach gleichen Algorithmen die interpolierten Bewegungsabschätzungen (Sender sendet Update bei zu großer Abweichung, sanfte Anpassung auf Empfängerseite zur Vermeidung von Sprüngen in der Anzeige*
- Maximale Signalverzögerungen ca. 100 ms bei menschlichen Akteuren

HLA – Aktuelle Entwicklungen

Neue Versionen der Standards (IEEE 1516 -2010)

- Neue Softwareanbindungen: nun auch DLL mit API (statt nur Source-Code-Libraries in C und Java -> Neukompilierung der Software bisher notwendig)

HLA Evolved (2010) (Verbesserte HLA-Version)

- Extended support for additional transportation (such as QoS, IPv6,...)
- Web Services (WSDL) support/API
- Extended XML support for FOM/SOM
- Fault tolerance support services
- Modular FOMs
- Update rate reduction
- Encoding helpers
- Standardized time representations

HLA – Umsetzung / Implementierungen

HLA – RTI ´s

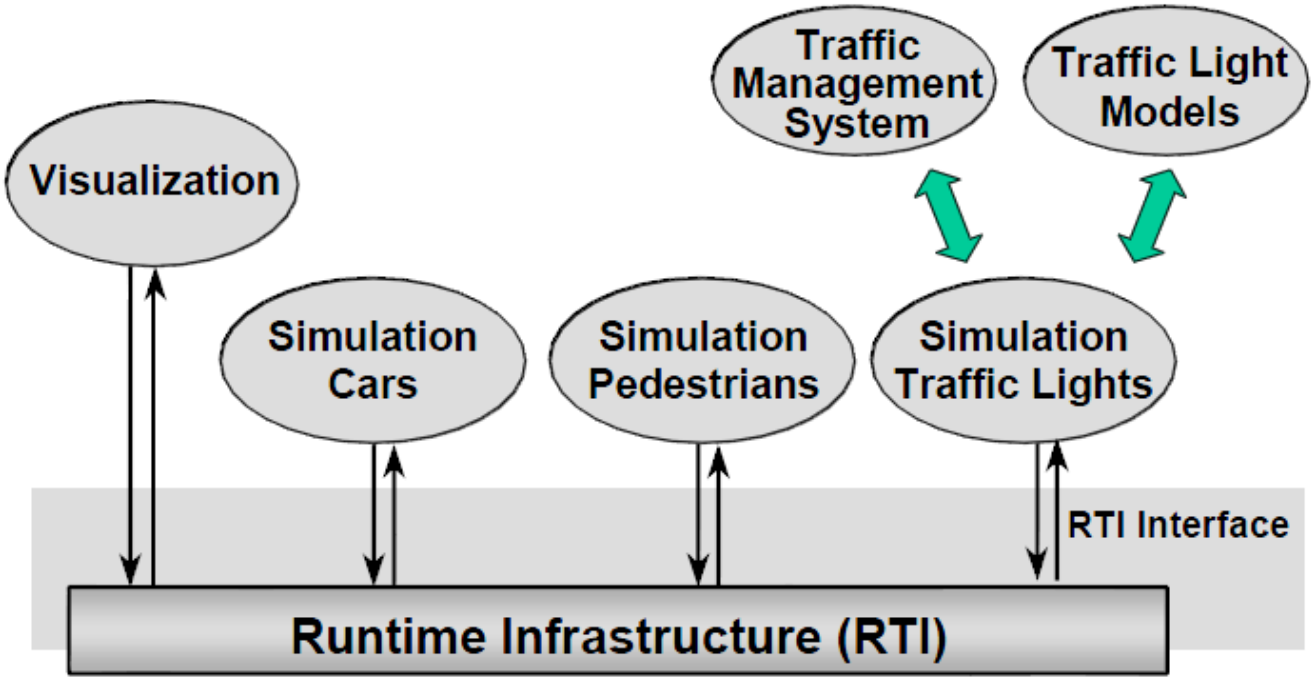
- zur Kopplung einer Federation ist i.d.R. EINE Basis-RTI eines Herstellers notwendig,
- Aufgrund nicht vollständig definierter interner Protokolle und Kommunikationskanäle (zwecks Fexlibilität und Performance sind RTI nicht zwingend kompatibel -> Test bzw. Prüfungen notwendig)

- **Verschiedene Hersteller** (siehe Liste unter
  - http://en.wikipedia.org/wiki/Run-Time_Infrastructure
  - US-Government –Implementierungen (meist nur Militär-intern)
  - Pitch Schweden – für auch zivile Anwendungen
    - sehr leistungsfähige und schnelle RTI
    - gut dokumentiert (100 Seiten Handbuch) unter http://www.pitch.se/hlatutorial

HLA - Zivile Anwendungen

Eine HLA-Straßenverkehrssimulation

- Abbildung der Verkehrskomponenten (vgl. Grafik) als eigenständige SLX-Modelle
- SLX wurde erweitert um eine SLX-RTI-Schnittstelle



Fazit / Zusammenfassung

Verteilte Simulation allgemein

- Fokusverlagerung von reiner Performanceoptimierung auf verteilte Simulation zur Nachnutzung komplexer, vorhandener Simulationsmodelle
- Der Speedup hat sich in den letzten 10 Jahren eher generell verringert, wegen der stark gestiegenen Rechenperformance bei moderat gestiegenen Kommunikations-geschwindigkeiten (Latenzzeiten immer noch im ms-Bereich)
- Synchronisation noch das größte Problem bei paralleler Simulation
- effizienter sind Hybridsimulationen über Gesamtmodelle
- GPU-Einsatz in Teilgebieten (MC) sehr interessant, aber noch nicht universell

Technologien zur Verteilten Simulation und Standards

- (immer noch) stark militärisch getrieben
- einige positive Anwendungsbeispiele im zivilen Bereich
- HLA als aktuellste Technologie teilweise noch etwas starr bei der Anwendung (kein Problem bei großen Militärmodellen, kritisch bei kleinen Firmenmodellen)

Gesamtfazit: Sowohl technologisch wie auch anwendungstechnisch sind verteilte Simulationen ein sehr interessanter Ansatz. Im Detail sind jedoch einige Fallstricke und Performancefallen zu beachten bzw. vorab auszutesten (sonst ggf. Speedup < 1 !!)

Literatur / Quellen zur verteilten Simulation

Quellen

- Fujimoto, R. M. (1999). Parallel and Distributed Simulation Systems, Wiley Interscience.
- Jefferson, D. R. (1990). Virtual Time II: Storage Management in distributed Simulation. Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing: 75-89.
- IEEE Std 1278.1-1995 (1995). IEEE Standard for Distributed Interactive Simulation – Application Protocols. New York, NY, Institute of Electrical and Electronics Engineers, Inc.
- Defense Modeling and Simulation Office (DMSO). 1998. The High Level Architecture
- U. Klein und S. Straßburger. Die High Level Architecture: Anforderungen an interoperable und wiederverwendbare Simulationen am Beispiel von Verkehrs- und Infrastruktursimulationen. In Simulationstechnik, Tagungsband 11. Symposium in Dortmund, November 1997, ed. A. Kuhn und S. Wenzel. Seiten 529-534, Vieweg 1997.